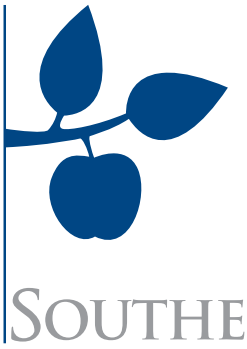
Robotics and Computer Vision 2 for UAS



University of Southern Denmark

Miniproject two – Traffic analysis

Frederik Mazur Andersen
fande14@student.sdu.dk

Kristian Møller Hansen
khans13@student.sdu.dk

Olliver Ordell
olord13@student.sdu.dk

Hand-in date: March 22th, 2018

Contents

1 Exercise 1

The video selected for this project is called `2015_06_27_1630_Krydset Motorvejsafkørsel 52` and can be viewed **here**.

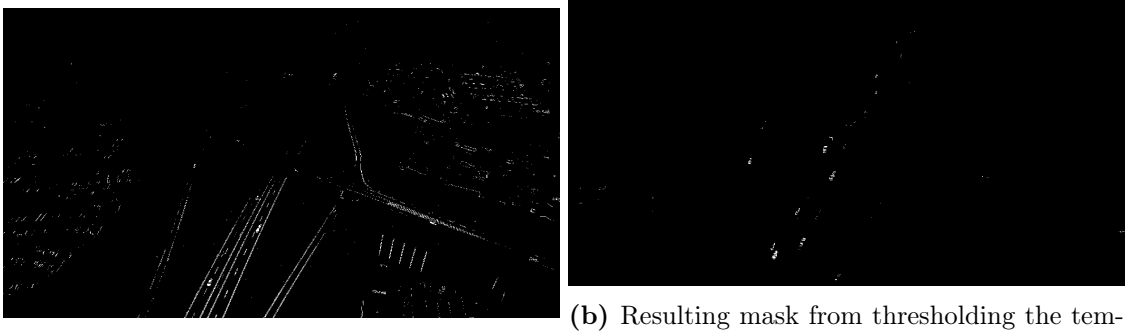
2 Exercises 2 & 5

For detecting moving objects background segmentation was needed. After removing the background a method to get the blobs and their position was needed for getting trajectories. Multiple methods was tried and a final method was chosen. This section will describe the methods tried and their outcome.

2.1 Background segmentation

2.1.1 Running average and temporal difference

Taking the running average with a low weighting would give a background image with no cars, but also get quite blurred. Then subtracting the difference over time based on new frames should give the moving objects. However it was found that it gave poor results as the image got too blurry and the missing stabilization of the video gave too much noise. A frame from this method is shown in figure 1.



(a) Resulting mask from thresholding the running average frame.

(b) Resulting mask from thresholding the temporal difference. This frame looks alright, but at certain points in the video the noise gets much worse.

Figure 1: Showing the resulting masks when thresholding the temporal difference between last and current frame and when thresholding the running average.

2.1.2 Neural network

A pretrained neural network from google called `mobilenet-ssd` was used[1]. It is pretrained, but as was found out, it was pretrained on images of cars from the front, side and back. Thus it didn't give results when the cars was seen from above a long distance away. We thought about training our own network, but didn't have access to easy training data and time constraints meant it was choosen to explore other methods.

2.1.3 Optical flow

Optical flow was tried, with back-confirmation of points, to detect moving features by using the built in optical flow methods from OpenCV. This gave good results for tracking cars,

but it only picked up few of the cars. We didn't manage to find out exactly why, but our leading theory is that it works on grey-scale images and especially cars with same colour nuance as the road had difficulties getting picked up. When it did pick up a car it kept tracking it quite accurate due to the back-confirmation. The back-confirmation is essential just going back from the points found to the previous frame and confirming that the points it find that way matches the original points. So it finds points both from previous frame to current frame and from current frame to previous frame. A frame from this is shown in figure 2.

Figure 2: The result of using optical flow with back-confirmation. Note how the white cars gets picked up, but the cars with same colour as the road do not get tracked.

2.1.4 MOG2

The solution that worked best was using the built-in OpenCV background-subtractor called MOG2. It was used with the parameters of `DetectShadows=False, History=50, VarThreshold=30` for best performance based on trial and error. Using these options removed the noise for the small movements the camera did. Besides that, the morphology operations of opening and closing was applied with a kernel size of 3 and 4 to remove small noise sources and connect cars together into single blobs. The result of using this is shown in figure 3.

To remove the last noise and get the blobs from the found moving objects the OpenCV method of `ConnectedComponents` was used with a threshold of area size of 38. This finds all white pixels that is connected and returns it as a blob. Then all blobs that have an area less than 38 is thrown out to eliminate the last noise in the frame. This gives enough detections for applying and correcting the implemented Kalman filter.

2.2 Tracking

To track the position an algorithm was needed to ensure that the right Kalman filter received the correct detected position. Otherwise the Kalman filter could swap detections

Figure 3: The result of using the MOG2 algorithm from OpenCV. Notice that there is still a little noise even after using morphology operations

with other blobs and thus not work correctly. The algorithm used for this was Nearest Neighbour. At first an attempt to use optical flow was attempted, but that didn't work out and gave bad results. To limit how far away a good neighbour could be, a euclidean distance of 35 was chosen as a limit. The algorithm goes though the list of Kalman filters and for each choose the nearest neighbour. This isn't optimal as it doesn't take into account if a later Kalman filter would be even closer to the nearest neighbor for the current Kalman filter. This is something that could be improved in the future.

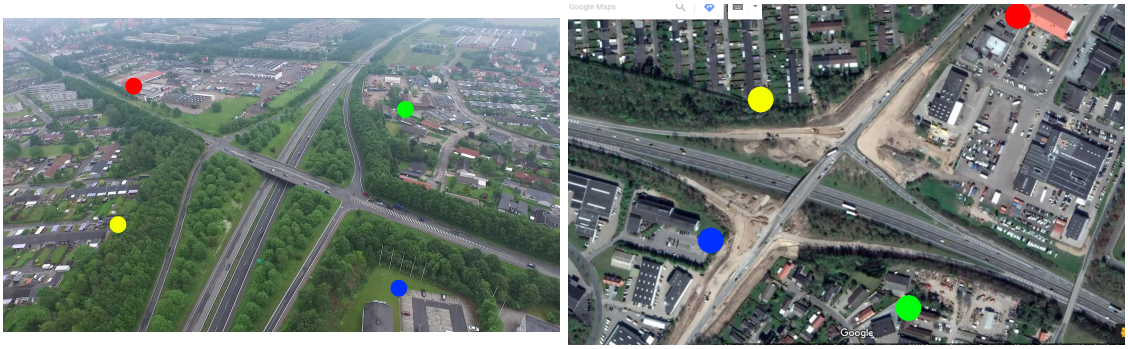
This gave positive results, but still don't connect the right detection to the correct Kalman filter 100% of the time. Especially when two cars meet at the bridge intersection it sometimes get confused. To future eliminate the pairing of wrong detections with filters, the Kalman filters implemented some optimization and limiting checks. these are described in section 5.5.

3 Exercise 3

On figure 4a four reference points are shown on top of a frame in the video. These points are then to be matched in Google Maps to be able to make a perspective transform, and see the video up top. The four corresponding points from Google Maps can be seen in figure 4b and table 1.

	Point 1	Point 2	Point 3	Point 4
Video	(450, 235)	(1385, 312)	(1362, 928)	(396, 711)
Google Maps	(846, 20)	(639, 573)	(266, 442)	(361, 178)

Table 1: Set of points used for homography. Points drawn in on figure 4



(a) Four reference points selected from one frame of the selected video

(b) Corresponding four reference points from Google Maps viewed from the top

Figure 4: Comparison of four selected reference point from one frame of the video, and the corresponding four points shown in Google Maps. Note that for (a), the top-left point is (b)'s top-right, and so on.

4 Exercise 4

In order to do the perspective transformation on the images, the OpenCV function `cv2.findHomography()` is used to find the homography matrix. This is then used in combination with the function `cv2.warpPerspective()` to find the new transformed image view from up top. A single frame of this cam be seen in figure 5

Figure 5: The result of using warpTransform on an image from the video. The image gets very distorted once you get far away from the bridge.

When making the perspective transform, there are a few assumptions that follows along the process. First of all there might be some human errors when selection the points, in that the matching points might not be in the exact same position. Furthermore the image acquired from Google Maps is assumed to be taken directly from above, which might not be the real case. And finally, the ground is assumed to be completely flat, which is definitely not the real case. These errors will all have an effect on the end-result of showing the speed of the moving cars.

When using this transform, it is possible to approximate the actual distance on the ground, by measuring the distance pr pixel in the image. This was done by measuring the the distance between two of the reference points on Google Maps. Then, the approximate

velocity of cars in the image can be calculated, using equation (1), once the pixel velocity from the Kalman filter is obtained.

$$\frac{\text{km}}{\text{h}} = \frac{\text{m}}{\text{pixel}} \cdot \frac{\text{pixel}}{\text{s}} \cdot 3.6 \tag{1}$$

5 Exercise 6

The idea behind using a Kalman filter to track objects is, that you can associate the state of the filter to the objects in the scene. And when it is not possible to find the object anymore because it was occluded or filtered out in the detection algorithm, then the Kalman filter can try to estimate where it thinks the object is based on its previous states and a model of the system. This means that you need a filter for every object that needs to be tracked in the scene, which is why it was implemented as a class.

5.1 Model

The model is based on Newton's laws of motion, where the state of the system contains the positions and velocities in the x- and y directions. There is no input to the system, because the position of an object is all that is known about it. The system is able to change the velocity of the state through the uncertainty in the covariance matrix, P, and the updates that come from consecutive correction steps. The filter is able to deduce that it is more likely that the state has a different velocity when the position keeps changing. The model can be seen in equations (2) through (5).

$$\vec{x}_k = F\vec{x}_{k-1} + \vec{w}_{k-1} \tag{2}$$

$$\begin{bmatrix} x_k \\ y_k \\ \dot{x}_k \\ \dot{y}_k \end{bmatrix} = \begin{bmatrix} 1 & 0 & dt & 0 \\ 0 & 1 & 0 & dt \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{k-1} \\ y_{k-1} \\ \dot{x}_{k-1} \\ \dot{y}_{k-1} \end{bmatrix} + \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \end{bmatrix} \tag{3}$$

$$\vec{y}_k = H\vec{x}_k + \vec{v} \tag{4}$$

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix} + \begin{bmatrix} v_0 \\ v_1 \end{bmatrix} \tag{5}$$

It is assumed that the errors associated with the process and the measurements have zero-mean and each individual term in the error vectors are uncorrelated. The process error has covariance matrix Q and the measurement error has covariance matrix R (eq. (6) & (7)).

$$\vec{w} \sim N_4(0, Q) \tag{6}$$

$$\vec{v} \sim N_2(0, R) \tag{7}$$

5.2 Implementation

The Kalman equations are separated into two parts; one for the prediction step (eq. (8) & (9)) and one for the correction step (eq. (10) through (12)). The prediction step only

consists of the process equation and an update of the covariance matrix P based on the multiplication with F. Every time a prediction is made, uncertainty is injected into the system by adding Q in the calculation of P.

$$\hat{\vec{x}}_k^- = F\hat{\vec{x}}_{k-1}^+ \tag{8}$$
$$P_k^- = FP_{k-1}^+F^T + Q \tag{9}$$

The superscripts, $-$ and $+$, respectively denote estimates based on prediction alone and based on both prediction and correction. The correction step makes use of the result of the prediction, and uses it to compare with a measurement in order to arrive at a more accurate estimate. The weighting between the prediction and the measurement is governed by the kalman gain, K.

$$K_k = P_k^- H^T(HP_k^- H^T + R)^{-1} \tag{10}$$
$$\hat{\vec{x}}_k^+ = \hat{\vec{x}}_k^- + K_k(\vec{y}_k - H\hat{\vec{x}}_k^-) \tag{11}$$
$$P_k^+ = (I - K_kH)P_k^- \tag{12}$$

5.3 Expanded Model

We also tried to use a model that can describe the acceleration as well as the position and velocity. This requires a few larger matrices, but the rest of the code is the same. The expanded matrices can be seen in equation (13) & (14).

$$\begin{bmatrix} x_k \\ y_k \\ \dot{x}_k \\ \dot{y}_k \\ \ddot{x}_k \\ \ddot{y}_k \end{bmatrix} = \begin{bmatrix} 1 & 0 & dt & 0 & dt^2/2 & 0 \\ 0 & 1 & 0 & dt & 0 & dt^2/2 \\ 0 & 0 & 1 & 0 & dt & 0 \\ 0 & 0 & 0 & 1 & 0 & dt \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{k-1} \\ y_{k-1} \\ \dot{x}_{k-1} \\ \dot{y}_{k-1} \\ \ddot{x}_{k-1} \\ \ddot{y}_{k-1} \end{bmatrix} + \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \end{bmatrix} \tag{13}$$

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_k \\ y_k \\ \dot{x}_k \\ \dot{y}_k \\ \ddot{x}_k \\ \ddot{y}_k \end{bmatrix} + \begin{bmatrix} v_0 \\ v_1 \end{bmatrix} \tag{14}$$

5.4 Integration with Tracked Points

A kalman filter will be assigned to every one of the tracked points. This is done by iterating through all already created filters and assigning the closest point that is within some distance of the state of the kalman filter. If no filter can be assigned to a point, then a new one will be created. The tracking box that is drawn on the image will only be created after the filter has had 30 corrections. This is to reduce clutter from spurious points. All filters will self-destruct if they do not receive any corrections for 80 consecutive frames.

All the tracked points are transformed into the top-down view before they are fed to the kalman filters, so the real velocities can be calculated approximately. After the state has been updated, the positions are transformed back into the original perspective, so the markers can be drawn in the video frame.

5.5 Optimizations

The implementation has some issues with the filters jumping between tracked points that are close together, because of the distance-based way that the points and filters are assigned to each other. In order to detect when a filter jumps to the wrong object, the angle between the change-in-position vector from frame to frame and the velocity vector is calculated. If this angle is too great the filter is destroyed, based on the assumption that this does not constitute a realistic movement of a car. This is not an ideal solution to the problem, because it might discard some valid tracking points, that moved in this manner because of particularly bad noise in the background segmentation. But it does get rid of some of the bad cases of filters jumping around between cars and getting thrown off the road.

5.6 Discussion

Both of the Kalman filter models were tried but it is difficult to judge which one performs better, because we do not have a good metric to make the distinction other than what they look like on the surface. They are also quite sensitive to the parameters; things like the allowed distance when a filter is assigned to a point and how the error matrices are initialized, have a big effect on how much the filters jump around and lose the objects. It seems like the expanded model with the acceleration has an easier time changing its velocity, which is nice when the cars move around a lot, but it also means that it is more volatile; it sometimes veers off the road completely whereas the velocity model tends to stay on the road. To illustrate how well the tracking works, the trajectories over the first two minutes of the video for both models, have been plotted on the image from Google maps. These can be seen in figures 6 and 7.

Figure 6: This is the trajectory from the first model that only tracks position and velocity in the state. The single two brown and green trajectories that are outside of roads are actually seagulls that gets picked up by the tracker.

Figure 7: This is the trajectory from the second model that also keeps track of the acceleration in the state. Here you can see that some of the trajectories also veer off the road. The acceleration model is able to do curved trajectories when it can't get any corrections.

Both methods have trajectories that jump from one car to another. That mostly happens when two cars travel side by side. In both instances, the filters were able to track cars that drive under the bridge, but they also lose them from time to time. This is probably due to noise in the background subtraction. Interestingly, the filters have tracked the cars outside the image frame on the top-right ramp and on the lane heading out of the image. In the video, the image frame stops before the ramp meets the highway, but several cars have been tracked almost all the way until they need to merge in.

6 Exercise 7

After all the filters have updated their states, the velocities are drawn on the video frame. The result can be seen in figure 8

Figure 8: This is a sample of what the video frame looks like, with all the Kalman filter positions drawn

7 References

[1] chuanqi305. *MobileNet-SSD*. 2018. URL: `https://github.com/chuanqi305/MobileNet-SSD`.